

EECS 349 Homework 3

Problem 1

I chose to perform the experiments in this question with 10-fold cross validation. The combination of kernel and cost that is best for the data set seems to be when the kernel function is a radial basis function and the cost, C , is 10. As indicated by the results, many of the functions have statistical significance over the base line scheme. I used the paired student's t-test with both the percent correct and the root mean squared error to determine the best kernel and cost and statistical significance of these tests. The paired student's t-test seemed like the most appropriate statistical test because it ensures that each function and its percent correct and error are involved in determining significance. Inspecting the values from this also indicated that the percent error may follow some normal distribution, so this also lent itself to using a student's t-test.

```

Tester:      weka.experiment.PairedTTester
Analysing:   Percent_correct
Datasets:    1
Resultsets:  15
Confidence:  0.05 (two tailed)
Sorted by:   -
Date:        11/2/14 10:16 PM

```

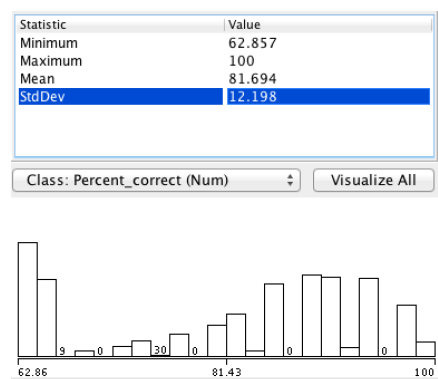
Dataset	(1) function	(2) func	(3) func	(4) func	(5) func	(6) func	(7) func	(8) func	(9) func	(10) func	(11) func	(12) func	(13) func	(14) func	(15) func
ionosphere	(1000 64.10 93.69 v 92.97 v 94.45 v 74.84 v 87.02 v 86.67 v 87.35 v 87.65 v 84.97 v 64.10 64.10 64.10 88.30 v 91.09 v														
	(v/ /*) (1/0/0) (1/0/0) (1/0/0) (1/0/0) (1/0/0) (1/0/0) (1/0/0) (1/0/0) (1/0/0) (1/0/0) (0/1/0) (0/1/0) (0/1/0) (1/0/0) (1/0/0)														

```

Keys:
(1) functions.LibSVM '-S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 0.01 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(2) functions.LibSVM '-S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 100.0 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(3) functions.LibSVM '-S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(4) functions.LibSVM '-S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 10.0 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(5) functions.LibSVM '-S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 0.1 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(6) functions.LibSVM '-S 0 -K 0 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(7) functions.LibSVM '-S 0 -K 0 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 0.1 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(8) functions.LibSVM '-S 0 -K 0 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 100.0 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(9) functions.LibSVM '-S 0 -K 0 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 10.0 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(10) functions.LibSVM '-S 0 -K 0 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 0.01 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(11) functions.LibSVM '-S 0 -K 1 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 0.01 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(12) functions.LibSVM '-S 0 -K 1 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 0.1 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(13) functions.LibSVM '-S 0 -K 1 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(14) functions.LibSVM '-S 0 -K 1 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 10.0 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172
(15) functions.LibSVM '-S 0 -K 1 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 100.0 -E 0.001 -P 0.1 -model /Users/GC -seed 1' 14172

```

In the figure above, we can see that function 4, which is a radial basis kernel function with a cost of 10, had the highest percentage of correct classifications and thus, is the best.



The output of the experiments are also attached in the zip as problem1.arff and problem1pairttest.txt

Problem 2

Part A

We can encode the emails (text documents) using the bag-of-words model. This model associates each word or phrase with an ID and its frequency count in an email. Using the IDs as indices, we can create a vector of simply frequency counts with which we can use in a support vector machine. So we

can map a document using this function to get the frequency of all the words in the document, and using this we can perform the inner product in a kernel.

Part B

We can perform an inner product on this mapping because by this mapping produces a vector of word or phrase frequencies with which we can simply take the inner product with and project into another space. If the mapping different in some way, taking an inner product might be more difficult but because of the preprocessing frequency count aspect of this mapping, we do not need to do anything to make this mapping capable of performing an inner product. As described in the article I referenced in part D, this mapping results in a dictionary or finite vector size of around 150,000 words due to the size of the english language.

Part C

This encoding for emails would be useful because support vector machines can recognize patterns in word frequencies of spam emails and classify them as spam or not spam. Words like “prize” and “win” will likely be words that have a higher frequency in spam emails and thus emails with words like these with word or phrase frequencies similar to training examples will be classified as spam. Using this we cannot only classify emails as spam, but we can also classify the chance of a word being a spam word.

Part D

I based my answer on the paper “Spam Classification using new kernel function in Support Vector Machine.” This paper is closely related to what I proposed because it similarly describes a dictionary like mapping for words in emails and using the frequency count of these words to create a vector with which to perform a inner product to classify.

Rakse, S. and Shukla, S. (2014). Spam Classification using new kernel function in Support Vector Machine. [online] Engineering Journals. Available at: <http://www.enggjournals.com/ijcse/doc/IJCSE10-02-05-131.pdf> [Accessed 3 Nov. 2014].

Linke: <http://www.enggjournals.com/ijcse/doc/IJCSE10-02-05-131.pdf>

Problem 3

Part A

It would take 10^{18} steps and it would take 10^9 seconds to run to completion.

Part B

To reduce the time it takes so that it is at least 10^7 faster, you must take a sample set of fewer than 4641 points. One method for selecting these points is selecting random samples, but this might result in an over abundance of one type of point over the other. While it would work, the separator might not be as good as if you picked a representative sample for every 10^3 samples, reducing the sample space to around 10^3 instead of 10^6 . Using either of these methods would improve the speed of the support vector machine and also get a close approximation (perhaps not maximal margin) of the SVM.

Part C

Assuming we took a random sample of 10^3 points and given that the machine processes one billion steps per second and the algorithm still takes a $O(N^3)$ to find the maximal margin, with a set of 10^3 points, it would take $(10^3)^3/10^9$, which is 1 second. Based on my calculation in part A, this is 10^9 faster than the naive approach of using all the data points.

I believe if we were to use a preprocessing step like selecting a representative point for every 10^3 points, we might need more information to compute the time it would take, depending on how long it would take to find a point that can represent 1000. Suffice to say, the first method works.

Part D

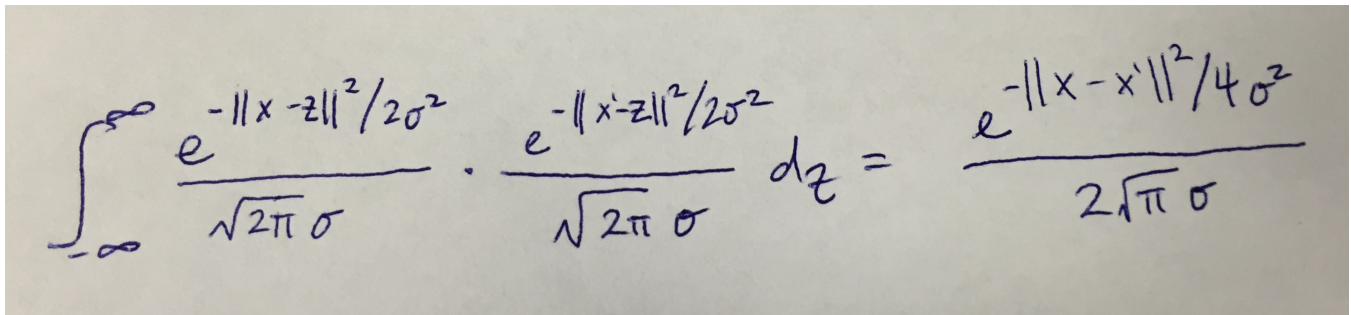
To determine the expected difference between the decision surface we learned in B and the decision surface we learned in A is statistically significant, we can take the difference between the classification accuracies and if this is within a margin of confidence, then we can assume that it works. Computing classification accuracy is as simple as comparing it against the training data. Thus, we can know how accurate each model is with some confidence based on the difference between the classification accuracy as a percentage.

Problem 4

A formula is a valid kernel if the function satisfies Mercer's Condition. Mercer's condition ensures that the kernel matrix will always be positive semi-definite, which ensures the training objective function is convex.

The Gaussian-Like formula provided is a valid kernel because it satisfies Mercer's Condition, which is the double integral of the kernel function with respect to x and x' . Since the exponential is always positive, and the double integral of this still remains positive, it means Mercer's Condition of positive semi-definite and thus means that it is a valid kernel. The exponential is always positive because there are no sign modifying parts of the function when you integrate it, and it is essentially a derivative form of an exponential kernel with the magnitude of the vectors instead of using variables ($k(x, x') = \exp(k_1(x, x'))$). The resultant is still a function that is always positive and thus satisfies Mercer's Condition.

Another approach to proving that this is a kernel is by proving a mapping can be presented such that $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$. We can do this by mapping x to a spherically symmetric Gaussian distribution centered at x in the Hilbert space L^2 . One dimension of this transformation is:

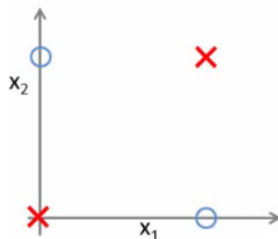

$$\int_{-\infty}^{\infty} \frac{e^{-\|x-z\|^2/2\sigma^2}}{\sqrt{2\pi}\sigma} \cdot \frac{e^{-\|x'-z\|^2/2\sigma^2}}{\sqrt{2\pi}\sigma} dz = \frac{e^{-\|x-x'\|^2/4\sigma^2}}{2\sqrt{\pi}\sigma}$$

Using the standard deviation of $\sigma/\sqrt{2}$ and scale the Gaussian distribution gets us a mapping $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$. We need to perform this re-scaling because of the L^2 from the Hilbert space. Through this transformation, we can see that the Gaussian-like function is a valid kernel.

Problem 5

Part A

A one-layer perceptron is not capable of building non-linear decision surfaces. Single-layer perceptrons cannot classify non-linear decision surfaces because of the structure of a perceptron. Perceptrons have input parameters represented as a real-valued vector and a weight vector with a weight for each input vector element. Taking the inner product of these two vectors with the addition of a constant bias for thresholding, results in a linear hyperplane in any hyperspace. Thus, one-layer perceptrons cannot implement functions like XOR as we discussed in class.



As you can see in the figure above, the XOR function is not separable by a single line.

Part B

Multi-layer perceptrons are capable of non-linear decision surfaces as long as the activation function is a sigmoid function. A single perceptron is simply an input vector, a weight vector, and a bias, and this results in a linear hyperplane in a space. By layering perceptrons in multiple layers and wiring them to other perceptrons, the result is a function with multiple linear separations, which are not dependent on a single set of weights resulting in different lines for classification. Thus, multi-layer perceptrons can implement functions like XOR.

With the figure in part A, you can see that it could be separated if multiple lines are used to separate the Os from the Xs. This can be accomplished with a multi-layer perceptron.

Part C

If the sigmoid function in a multi-layer perceptron were replaced with the following function,

$$output = sign\left(\frac{1}{1 + e^{-net}}\right)$$

$$sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{else} \end{cases}$$

it is essentially like replacing the sigmoid function with the activation function from a regular perceptron. The sign function is also not differentiable about the point 0 because this function contains sharp corners. Replacing the sigmoid with this sign function is counter productive to the concept of having a multi-layered perceptron with a sigmoid function because the point of the sigmoid function is to assist with back propagation error. Back propagation learning occurs by changing the weights of the perceptron. This can be accomplished by comparing the output result and the expected result and using the error adjusting the weights. Multi-layer perceptrons can find the correction we need to make using gradient decent to change each individual weight. However, gradient decent requires a differentiable function in order to back propagate error and learn, but because the new sign function essentially makes the sigmoid function a signum function, it is useless.

Problem 6

Part A

A Restricted Boltzmann Machine is a stochastic neural network that can learn probability over a set of its inputs. It does this by utilizing a set of visible units and a set of hidden units, which consist of a matrix of weights associated with the connection between the hidden and visible sections as well as a bias of weight offsets. Energy is defined between a pair of boolean vectors and is similar to the energy in a Hopfield network, which means that the Boltzmann machines are defined in terms of their energy function over the hidden and visible vectors.

Part B

Deep belief networks are related to Restricted Boltzmann Machines because you can construct a deep belief network by stacking RBMs. The network is basically composed of a series of unsupervised RBMs where each sub-network's hidden layer serves as the visible layer to the next, which leads to fast training. With this hidden and visible layer structure it means that we can train these networks layer by layer and conceivably get a machine to understand deep learning abstractions.

Problem 7

Part A

Kernel machines are non-parametric learning models, which means methods that allow the complexity of the solution to increase when more data is available. Kernel machines make weak assumptions on the form of the function to be learned. Non-parametric learning models encompass a wide range of models, including k-nearest neighbors algorithms, modern kernel machines, and multi-layer neural networks. Kernel machines are limited by the nature of shallow architectures and the local kernel. Kernel machines are a type 2 shallow architecture and in some cases, the kernel is used as a kind of template matcher, but it can also be used for other choices.

Part B

There are two limitations of Kernel Machines. The first limitation is a limitation of shallow architectures and less specific to only Kernel Machines. There are three types of shallow architectures, type 1, type 2, and type 3.

Type 1 is composed of a fixed preprocessing layer and a linear predictor. The preprocessing layer of this type is fixed and generally task-specific, hand-crafted and tailored to the problem. Type 2 consists of template matchers and linear predictors. Much like the first type, type 2 is composed of a preprocessing layer and a linear predictor. In the preprocessing layer, a vector of values resulting from the application of a kernel function on each training examples culminates in a template matching of the training samples. Type 3 are simple trainable basis functions and a linear predictor. The first layer of a type 3 shallow architecture consists of a basis function that is trainable through supervised learning.

These various types all describe persistent problem across most shallow architecture machines which is that they are limited in the efficiency of the representation of certain types of function families. With shallow architectures you trade depth for breadth. A pitfall for type 2 kernel machine like machines that rely on template matching is that they may require a large number of templates in order to avoid misclassification.

The second limitation is to do with local estimators and kernel. To obtain the consistency of non-parametric estimators we can control it using the locality of the kernel and have its statistical bias approach zero, but also the statistical variance is also driven by this and should approach zero. This becomes a limiting factor as complexity of tasks increases and this can make it impractical for the kernel machine to learn these functions.

Part C

Deep architectures are compositions of many layers of adaptive non-linear components, which basically means that they are a series of non-linear pieces with parameters that you can train in all of their levels. Deep architectures are generally more efficient than shallow architectures and thus allow the representation of wide variety of functions in a more compact form because they trade the breadth

of knowledge and learning for depth. Deep architectures differ from shallow architectures by trading breadth of learning for depth.

Part D

We can get around the limits of deep architectures by using simple gradient descent applied to conventional networks and layer-by-layer unsupervised learning by gradient descent. The unsupervised learning phase provides an initial configuration of parameters with which we can then run a gradient based supervised phase can be run on. This pairs each feed forward layer with a feed back layer which attempts to reconstruct the input of the layer from its output. This guarantees more information relation in the output layer. Since deep architectures are in their infancy, it is conceivable that we haven't come up with ways to circumvent the limitations of deep architectures just yet but we have two proven methods that work relatively well.

Problem 8

I ran my tests against a support vector machine with a radial basis function and a cost of 10. I ran these experiments against a three layer perceptron (single hidden layer). Due to time constraints, I compared a variety of different networks with different learning rates and different learning momenta. The network and a node for each input and two hidden nodes which connected to each of the inputs and these were connected to the outputs. I found that there wasn't a statistical difference between any of the variants of neural networks and that I used. I tested all combinations of networks with learning and momentum ranging from 0.2 to 0.8 incremented by 0.2. These all had the same wiring which might have been why the results were very similar for all the networks. Despite the variability in these networks, the radial basis SVM seemed to be the best. It would be interesting to run more trials with different wiring of neural networks and test learning and momentum rates.

```
Tester: weka.experiment.PairedTTester
Analysing: Percent_correct
Datasets: 1
Resultsets: 17
Confidence: 0.05 (two tailed)
Sorted by: -
Date: 11/3/14 12:07 PM
```

Dataset	(1) function	(2) funct	(3) funct	(4) funct	(5) funct	(6) funct	(7) funct	(8) funct	(9) funct	(10) func	(11) func	(12) func	(13) func	(14) func	(15) func	(16) func	(17) func
ionosphere	(10) 94.33	92.32	91.17	91.46 *	91.17	91.46 *	92.03	91.17	91.17	91.17	90.88	92.03	92.03	91.17	91.45	91.45	92.32
	(v/ /*)	(0/1/0)	(0/1/0)	(0/0/1)	(0/1/0)	(0/0/1)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)

Key:

```
(1) functions.LibSVM "-S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 10.0 -E 0.001 -P 0.1 -model /Users/GC -seed 1" 14172
(2) functions.MultilayerPerceptron "-L 0.2 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(3) functions.MultilayerPerceptron "-L 0.4 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(4) functions.MultilayerPerceptron "-L 0.6 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(5) functions.MultilayerPerceptron "-L 0.8 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(6) functions.MultilayerPerceptron "-L 0.2 -M 0.4 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(7) functions.MultilayerPerceptron "-L 0.4 -M 0.4 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(8) functions.MultilayerPerceptron "-L 0.6 -M 0.4 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(9) functions.MultilayerPerceptron "-L 0.8 -M 0.4 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(10) functions.MultilayerPerceptron "-L 0.2 -M 0.6 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(11) functions.MultilayerPerceptron "-L 0.4 -M 0.6 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(12) functions.MultilayerPerceptron "-L 0.6 -M 0.6 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(13) functions.MultilayerPerceptron "-L 0.8 -M 0.6 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(14) functions.MultilayerPerceptron "-L 0.2 -M 0.8 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(15) functions.MultilayerPerceptron "-L 0.4 -M 0.8 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(16) functions.MultilayerPerceptron "-L 0.6 -M 0.8 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
(17) functions.MultilayerPerceptron "-L 0.8 -M 0.8 -N 500 -V 0 -S 0 -E 20 -H a" -5990607817048210779
```

The output file for the experiment and the ttests are in the zip as question8.arff and question8pairedttest.txt.